



University
of Glasgow | School of
Computing Science

Honours Individual Project Dissertation

HUMAN MOTION ANALYSIS

Kshitiz Bisht
January 9, 2022

Abstract

To be able to develop a model that could predict the behaviour of the human by just analysing the motions can have lots of application. The goal of this project is to develop a machine learning model that could predict the motion of the human based on the given video.

First, we start by creating the database for the model to train on. The CSM (Character Studio Marker) files of different character were provided. CSM files are from the motion-capture device that contains only marker-position data, it contains information about the motion of the character. The csm files were used to create the animations of different types of motion in Autodesk 3ds Max. The animations were then recorded and then poses were extracted from those videos. The extracted poses served a training data. The model used in the project is a hybrid model of Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM).

The model was evaluated in two different approaches. First the model was trained and tested on both male and female videos. Second, the model was trained on male videos and tested on female videos. The model that was trained on first approach had an accuracy of almost 95% and model in second approach had an accuracy of almost 80% during validation. The model was then tested on real data. 4 people were selected and were asked to do the similar motion as that of the animations. This experiment was done to analyze how the model that was trained on virtual data will behave with real data.

Education Use Consent

Consent for educational reuse withheld. Do not distribute.

Contents

1	Introduction	1
1.1	Aims	1
2	Background	3
2.1	Pose estimation	3
2.1.1	Summary	4
2.2	Human action recognition	4
2.2.1	Summary	4
2.3	Using Synthetic data for machine learning	5
2.3.1	Summary	5
3	Requirements	6
3.1	Must Have	6
3.2	Should Have	6
3.3	Could Have	7
4	Design and Analysis	8
4.1	Programming Language and Machine Learning Modules	8
4.1.1	Python	8
4.1.2	Tensorflow and Keras	8
4.1.3	Scikit-Learn	8
4.2	Making dataset	9
4.2.1	Mixamo Characters	9
4.2.2	Metahumans	9
4.3	Pose estimation	10
4.3.1	OpenPose	10
4.4	Neural Network Architecture	11
4.4.1	Activation Functions	12
4.4.2	Backpropagation	13
4.4.3	Convolution Neural Network (CNN)	14
4.4.4	Recurrent Neural Network (LSTM)	15
4.4.5	Function of RNNs	16
4.4.6	CNN + LSTM model	19
5	Methodology	21
5.1	Given dataset	21
5.2	Overview of the framework	21
5.3	Animating motions in 3ds Max	22
5.3.1	Autodesk 3dsMax	22
5.3.2	Creating animations	22
5.4	Pose Extraction	23
5.5	Combining poses and making labels	24
5.6	Structure of model	25

6	Evaluation	27
6.1	Metrics used	28
6.1.1	Loss	28
6.1.2	Accuracy	28
6.1.3	precision	28
6.1.4	recall	29
6.1.5	F1 score	29
6.1.6	Confusion matrix	29
6.2	Training - Approach 1	30
6.2.1	Accuracy And Loss	30
6.2.2	Results	30
6.2.3	Summary of Approach - 1	31
6.3	Training - Approach 2	32
6.3.1	Accuracy And Loss	32
6.3.2	Results	33
6.4	Comparison with Baseline Model	33
6.5	Testing on Real-life data	34
7	Conclusion	36
7.1	Summary	36
7.2	Summary of contributions	36
7.3	Limitations	36
7.4	Future work	37
	Appendices	38
A	Appendices	38
A.1	Ethics information: Ethics consent form	38
A.2	Ethics information: Introduction and Debrief scripts	42
	Bibliography	45

1 | Introduction

Human motion analysis is something that everyone does but they don't realise it. Let us understand this by a simple example, if you see a person hopping while walking on the street, you can easily comprehend that the person is happy. This simple process of analysing the behavior of the person by just understanding the motion of the person is known as motion analysis.

Human motion analysis have many applications. For example surveillance, Martinez-Mascorro et al (2020) in Suspicious Behavior Detection on Shoplifting Cases for Crime Prevention by Using 3D Convolutional Neural Networks proposed a neural network that identifies suspicious behaviour of an individual before committing an offensive act. The model reached an accuracy 75% in predicting suspicious behaviour. This work can further be extended to use other areas such as bank, mall or other areas that are vulnerable to crimes.

The idea is good but the real question is how the system or the model will be able to identify suspicious behaviour?. Making a model to predict something based on the given data requires dataset so that the model is able to train and learn. These datasets are mostly available publicly, one can just simply download the dataset and use it but what if one requires some real world data. When dealing with the motion analysis real world data there are two major concerns. First, privacy of the users is the number one concern as the data may have some sensitive information about the user. e.g., if making a system that separates the important e-mail such as bills, works, receipts from other e-mails such as ads, spams. The system needs sample emails to train and classify but because of data protection legislature we need to consider the user's consent and privacy. Similarly, if a system has to analyse human motion it first needs a sample of human motion videos which requires the consent of that person. Second, is the limited number of dataset for human motion analysis. To tackle these problem the project used synthetic data on virtual character to maintain the privacy of the users and make a dataset of different motions.

1.1 Aims

Considering everything in motivation we will make an attempt to make a machine learning model for human motion analysis. Although the ultimate goal of the project is to make a model understand human motion, it can be broken in to multiple steps

- **Creating a dataset of different types of motions with virtual characters using 3dsmax.** Every machine learning model need the dataset to train so that it can learn pattern and sequences involved in the data but there is no large dataset available for human motion. Therefore, there is a need develop mechanism to synthesise data. This data was created with the help of the character markup files (.csm) in 3ds max. The dataset contained the animation videos of 2 virtual characters. The different motions involved in the dataset were knocking, walking, lifting and throwing.
- **Designing and implementing a deep learning model that will be trained on data and predict the motion.** The model would be used to analyse the motion videos by classifying each of the frame of the video. It would recognize the activity as the maximum number of frame label in a particular class.

- **Analysing and evaluating the performance of the model.** After training the model on the data, the performance of the model is measured using different metrics. The performance of the model is also compared with the baseline model.
- **Performance of model on real data.** This experiment was done to analyse how the model trained on the simulated data will behave with real data to see how this translates to real world scenario.

2 | Background

This chapter will deal with all the related work done in the field of human motion analysis.

2.1 Pose estimation

Pose estimation means predicting the poses of human in a video or an image. Human pose estimation is generally a computer vision task that represents the different poses of the body in form of coordinates of keypoints. The keypoints includes right shoulder, left shoulder, nose or any other part of the body. Pose estimation is a very important task as it can help a machine to predict or understand human behavior. At the current state-of-the-art pose estimation mostly includes machine learning models, especially convolutional neural network(CNN), to extract human poses. CNN is commonly used because of its capability to process images. There are many different methods for pose estimation but all of them can be grouped into two general categories, Bottom-up and Top-down approach.

Top-down This approach first detects the human in an image or a video and then performs pose estimation to predict the pose of the human in the bounded box. Papandreou et la. 2017 proposed pose estimation with top-down approach using Faster-RCNN to detect the human in an image or video, predict heatmap and offsets using full convolutional ResNet (CNN model) to obtain the location of the joints in human body. Fang et la. 2018 introduced regional multi-person pose estimation in which the bounded box containing the human was fed to symmetric STN and SPPE module that outputs pose proposals which were refined by parametric pose NMS to output pose. Ning et la. 2019 modified the Cascaded Pyramid Networks(CPN) to perform pose estimation on the bounded boxes obtained from the detector.

Bottom-up This approach estimates all body joints of all the people in the image or video and then group them to form pose for each person using different types of joint association techniques. One of the popular method using Bottom-up approach is Deepcut, proposed by Pishchulin et la. 2016, it does the tasks of detection and pose estimation, it jointly partitioned and labelled the joints with the help of CNN. Each part had a label of its class it belonged, then the integer linear program (ILP) would assign these joints to the person it belonged. But the IPL used by deepcut is hard to approximate which results in high runtime. Insafutdinov1 et la. 2016, introduced improved Deepcut which used branch and cut algorithm, this resulted in 4-5 times reduction in runtime and increased in accuracy of the poses. The approach used enhances part detector model, to predict the location of a body part precisely and image-conditioned pairwise terms for pose estimation. Cao et la. 2016 uses part affinity fields(PAF) to estimate pose. The method predicts confidence map and PAFs and then uses a greedy interface to produce poses of all people in the input. Later, Li et la. 2018 proposes a new approach for pose estimation with bounding box constraint, which combines both bottom-up and top-down methods. The method estimates the confidence map the direction fields with the help of CNN model which has been trained to learn confidence map of joints and connection relationships between joints, then connects the joints and form the full pose within the bounding box. Geng et la. 2021 introduced pose estimation using disentangled keypoints regression which estimated pose at every pixel, also

known as center pixel, by activating them and learn representation from them with the help of multi-branch structure for separate regression.

2.1.1 Summary

The bottom-up approach identifies all the keypoints and then joins them to form a pose for the person. On the other hand, top-down approach first detects the human and then performs pose estimation. Top-down approach though being more accurate is more costly because of extra person detection phase. The accuracy of this approach depends highly on detection phase because pose estimator is usually sensitive to the bounded boxes detected and also this approach have difficulties in detecting when two person in an image overlaps. The runtime of top-down approach increases as the number of people in the input increase. Bottom-up approach even though is faster in predicting keypoints takes time to assemble all the keypoints to form a pose. Sometimes during the process of assembling keypoints wrong keypoints may get connected due to truncation error which can result in wrong pose. Keeping all these points in mind the project uses one of the popular bottom-up approach library called OpenPose.

2.2 Human action recognition

Baccouche et la. (2011) in Sequential Deep Learning for Human Action Recognition proposed a fully automated deep learning model that can learn to classify human actions with any prior knowledge. They achieved this in two steps, first they extended the CNN model that learn extracts spatio-temporal features and second, they then train a RNN model to learn the sequence of features at each timestep. Wu et la. (2015) in their Modeling Spatial-Temporal Clues in a Hybrid Deep Learning Framework for Video Classification used the similar approach where they used CNN to extract features and RNN to predict time-series data but instead of using one CNN model they used two CNN model to extract spatial and short-term motion, then theses two CNN combined for classification. Later in the same year Sun et la. (2015) in Human Action Recognition using Factorized Spatio-Temporal Convolutional Networks proposed a new method to deal with 3D data for action recognition. They made a new network that was stack of spatial convolutional layers of 3d followed by 2d which was followed by 1d, the paper mentioned that the model was able to CNN based methods of that time. Donahue et la. (2016) in Long-term Recurrent Convolutional Networks for Visual Recognition and Description introduced a model that combined ability of a convolutional layer along with long-range temporal recursion. The model combines the ability of the CNN to extract the features and the ability of LSTM to work with time-series data. The paper also compares the performace of the model using factored layer which was proposed by Sun et la(2015). The horizon for the use of CNN along with the LSTM for human activity recognition broadened when Gammulle et la(2017) proposed four different fusion methods for combining CNN and LSTM model and these methods outperform the models of that time. Later, Ullah et la(2017) in Action Recognition in Video Sequences using Deep Bi-Directional LSTM With CNN Features proposed the use of CNN with deep bi-directional LSTM. It was called bi-directional as layers were stacked in forward and backward passes. This model was able to learn sequences from a very long video.

2.2.1 Summary

All the paper mentioned in the earlier section introduces new was in the field of human activity recognition but they have one thing in common, which is the use of CNN for extracting the features and then the use of LSTM for analysing how the features are changing in the input data at each time step. Keeping this in mind the model used by the projects combines the CNN and LSTM to take the best of both worlds.

2.3 Using Synthetic data for machine learning

The machine or deep learning has lots of potential to improve and be used in the world but these improvements are hindered by the limited data available for research purposes. Getting a dataset for a model to train could be a issue as it may contain some sensitive information about an individual. This problem is tackled by using synthetic data. Synthetic data is a type of data that mimics the real data by using the underlying statistics involved in the real data. Most common ways of producing synthetic data includes using neural networks such as Generative adversarial network or using computer graphics or gaming engines. The use of synthetic data in medical purposes is increasing, Rankin et la (2020) in Reliability of Supervised Machine Learning Using Synthetic Data in Health Care: Model to Preserve Privacy for Data Sharing used synthetic data generator to train five supervised models. The synthetic data was generated with classification and regression trees, parametric, and Bayesian network methods. The accuracy of the model was measured using the real data. There was a decrease in accuracy when the model trained on synthetic data was tested on real data. Another use of synthetic data in medical purposes was done by Chen et la (2021) in Synthetic data in machine learning for medicine and healthcare where, they used GANs to produce synthetic images by training a GAN with 10,000 real images of each type. They then compared the performance of the model with another model which was trained using both real and synthetic data.

The use of synthetic data does not end in only medical purposes, its use is also extended in human action recognition. Puig et la(2018) in VirtualHome: Simulating Household Activities via Programs developed a simulation program where they simulated simple household activities using Unity3D gaming engine. This enabled them to create a large video dataset. Another simulation application was created by Hwang et la (2020) in ELDERSIM: a synthetic data generation platform for human action recognition in eldercare applications where they developed a action simulation called Eldersim where it generated synthetic data on elders' daily life. This platform was developed using Unreal Engine as rendering platform and Autodesk Maya for animations and modelling. The animation used the motion capture file which was obtained from the elders. Varol et la (2021) in Synthetic Humans for Action Recognition from Unseen Viewpoints proposed a new method to generate synthetic action videos with labels by extracting the 3D dynamics of a human from a single view and combining it random components such as viewpoints or clothing to create a large set of training data.

2.3.1 Summary

The use of synthetic data is increasing due to limited availability or accessibility of dataset to preserve sensitive data of an individual. Researchers are tackling this issue by using neural network or gaming engine to create synthetic data. This projects also tackles the same issue by making a dataset of different motions with the help of virtual characters from Mixamo and using a modeling software called Autodesk 3dsMax

3 | Requirements

The chapter will deal with the requirements of the model that were decided throughout the course of meetings. The requirements are provided using the MoSCow method, this method was chosen to in order to prioritize the requirements of the project.

3.1 Must Have

These were the requirements that were on the top of priority list.

1. **A complete dataset of motions.** In order to train the model to classify types of motion it required the complete dataset of different types of motions. To do this character markup files (.csm) files of different characters were provided. These files were to be used to animate the motions (Further explained). The zipped folder of csm files contained different types of motions which were throwing (an action of throwing something), walking, lifting(pressing the button of the lift) and knocking.
2. **Animation should be done using 3ds Max and Mixamo characters.** Keeping section 2.3 in mind, making dataset using 3ds Max was considered. The csm files contained the data about the motion of the character which could be animated with the help of 3ds Max. The animations were to be created using a female and male characters. To make the dataset more realistic characters from Mixamo was chosen to be animated.
3. **Pose extraction on the videos.** After making dataset, the next step is to extract poses from the video. The pose extraction would be done on each frame of the video resulting in an array of sequential poses. These poses would then be combined to form the training data for the model. For pose extraction the project used openPose.
4. **Motion analysis model** - A machine learning model, using the current state-of-the-art methods, to identify the different types of motions. Keeping in mind section 2.3, the model should be able to learn the different features and also recognize the pattern or sequence in the training data. To achieve this aim, a hybrid model of CNN and LSTM was selected. This model will be able to extract the features using the CNN and will also be able to recognise the sequence in the data using the LSTM.
5. **Evaluation of the model** - The model will be trained in two different approaches. First approach, the model will be trained on both male and female motion videos and then would be evaluated on different metrics such as accuracy, loss, f1 scores, precision etc using male and female videos. Second approach, the model would be trained on only male video and then will be evaluated on same metrics but using female videos.

3.2 Should Have

This section will deal with the requirements that the project should have.

1. **Input data for Model** - Since the model is expected to compute on the time-series data (videos), the sequence of frames is important. The poses extracted should be combined in way that would preserve the sequence information of animations. The model will accept

the data frame by frame. The labels of the training set should be created keeping the mind the order of poses. To achieve this aim, the order in which the poses were combined was noted.

2. **Comparison with baseline** - The model proposed by the project should be compared with the baseline model to show how the new model is better than the traditional baseline model. To achieve this aim, a CNN model was selected for baseline.

3.3 Could Have

This section will discuss about the requirement that would be good to have in project.

1. **A machine learning model for gender classification.** This was basically considered to get familiar with the CNN models. The model was made to classify male and female videos.
2. **Real-data evaluation** - The model is to be trained on the virtual videos of virtual characters. But what would happen if the model is tested on a video of real-person, how will the model behave. This aim was chosen as topic to be discussed as how a machine learning model will bridge the gap between virtual and real life. To achieve this aim, an experiment was conducted where the videos of 4 different people doing the similar actions of knocking, lifting, throwing and walking was recorded. The model was then tested on these videos to how the model will behave with real data.

4 | Design and Analysis

This chapter will discuss all the given files, technologies considered and explanation of each component of the project in detail.

4.1 Programming Language and Machine Learning Modules

4.1.1 Python

- The project is written in Python. It is a high-level programming language. It is called high level language because it allows developer to read and write code in languages that are close to natural language. Python was chose for this project because it is easy to read and concise. Python is very to understand by human which makes the life of programmers easier when it comes to understanding other programmer's code. Python provides numerous libraries to be able to use for machine learning or deep learning, some of the popular libraries includes Tensorflow, keras, Scikit-learn and pytorch.

4.1.2 Tensorflow and Keras

Tensorflow is am end-to-end, open-source machine learning platform. Tensorflow is an ecosystem for number of tool, libraries, modules and other resouces. Tensorflow have several different abilities:-

1. Computing differentiable expressions.
2. Ability to scale the computation power to clusters of GPUs.
3. Executing tensor operations on CPU or GPU.
4. Exporting programs to external services like servers.

On the other hand, Keras is the neural network module of Tensorflow. It is API for deep learning written in Python. Keras is simple, flexible anad powerful library to use as it reduces programmer's cognitive load by providing simple and easy to use libraries. It is a very popular module as it is also used by YouTube and NASA. This project uses keras for making the machine learning model.

4.1.3 Scikit-Learn

Similar to keras, scikit-learn provides numerous supervised and unsupervised algorithms for machine-learning. It is actually a free library written in Python. This project uses this module mostly for the evaluation purposes. Many modules of Scikit-Learn were used for evaluations of the model after training.

4.2 Making dataset

For making a dataset for the model, the animations videos were required. To create animations, the project required characters to animate. The subsections below explains how characters were chosen.

4.2.1 Mixamo Characters

The project required characters for creating animations. The characters should be realistic as the model would later be tested on real-world data. For this purpose, The characters were taken from Mixamo. It is a 3D computer graphics technology company that provides animated characters. Mixamo provides numerous different characters that comes fully textured and rigged i.e., they come ready to animate so they can be used in any projects. Another specific reason for choosing Mixamo was that the characters are compatible with Autodesk 3ds Max.



Figure 4.1: The figure above shows the picture of male and female characters that were chosen for animations.

4.2.2 Metahumans

To make the project more realistic Metahuman was considered as an idea for choosing characters. Metahumans are provided by UnrealEngine, they are high-quality digital humans who looks as real as a normal real-life human (see figure 4.2). The metahumans are created using MetaHuman Creator. It is a free cloud-based app that is used to create realistic digital humans with different hairstyles and clothing. The main idea to choose metahumans was because different metahuman can be created easily and they come ready to animate which could have provided diversity in dataset. But the project used Mixamo's characters for the animation because there were several complications to animate the metahumans. The complications were, Metahumans could only be exported to Maya and UnrealEngine but the project was using 3ds Max so it was not possible to animated them. Making the animations from 3ds Max and then exporting them to the UnrealEngine was seen as an alternative route but due to the difference in skeleton of the characters in 3ds Max and UnrealEngine, the animations had to be mapped for it to work properly which could have taken a lot of time and effort to just animate one motion.



Figure 4.2: The picture above is of Metahuman created using Unreal Engine

4.3 Pose estimation

Pose estimation is a computer vision technique which is used to comprehend the position and orientation of the object in an image. It basically gives us a mean to track the object by locating the keypoints of the object. For the purpose of pose estimation openPose was used.

4.3.1 OpenPose

OpenPose is a real-time multiple-person detection library developed by researchers at Carnegie Mellon University. It is able to jointly detect the human body, foot, hand and facial keypoints of an image. It is able to detect 135 keypoints. Openpose is written in C++ and Caffe.

OpenPose architecture This section will explain the working of OpenPose.

1. An image is fed to CNN that produces two different outputs also known as "**two-branched multi-stage**" CNN. The "multi-stage" is used to increase the depth of the network means that at every stage, network is stacked on top of other.
2. The top branch is responsible for predicting the confidence maps of different body parts e.g., left shoulder, right knee, left eye and other. The other branch or the bottom branch is responsible for predicting affinity fields, this shows the degree of association between different parts of the body.
3. In the end, the affinity fields and confidence maps are computed by the greedy algorithm to output 2D keypoints.

Confidence Maps Confidence maps are produced by the top branch of the model. It is 2D representation representing the possibility of a particular body part in a given pixel. It is defined as

$$S = (S_1, S_2, S_3, \dots, S_J)$$

$$S_j \in \mathbb{R}^{w \times h}$$

$$j \in 1 \dots J$$

where J is the number of body part.

Part Affinity Fields Part affinity fields plays an important part in the multi-person pose detection because it helps in distinguishing between different people when they are grouped together. It represents the chances that the particular body part belongs to a person.

$$L = (L_1, L_2, L_3, \dots, L_c)$$

$$L_c \in \mathbb{R}^{w \times h \times 2}$$

$$c \in 1 \dots C$$

where C represents the number of limbs.

4.4 Neural Network Architecture

Neural network are networks made of neurons, in case of machine learning it is made of artificial neuron. These neural network can take multiple inputs to produce output. The components of neural network are:-

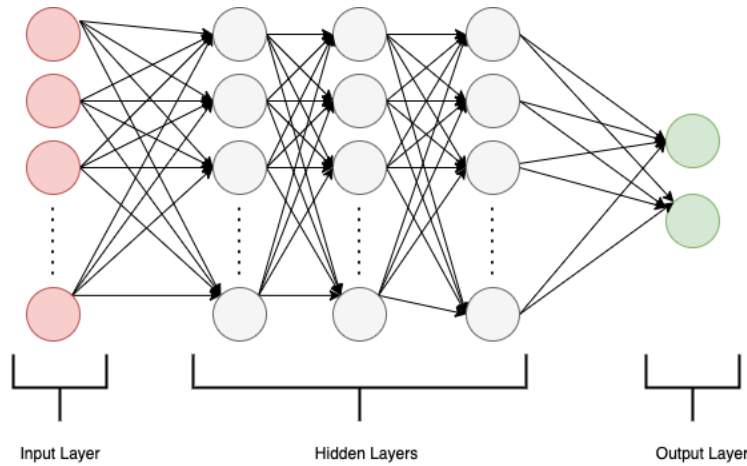


Figure 4.3: Neural Network model

1. **Input Layer, Neurons and Weights** - The fig 4.3 shows the neural network that is divided into three parts Input, hidden and output layer. The red circles in the figures forms the input layer, which is responsible for receiving the input information. Each red circle is called neuron. Each neuron is connected to another neuron of next layer forming a fully connected network. The connection between each neuron has a weight which are assigned to each neuron based on its importance in accordance to input.
2. **Hidden Layer** - This is the next layer after input layer. This layer is responsible for main computation of the information. The information from input node is multiplied with the respective weights and added together along with the bias unit. This forms a linear equation which is passed into activation function. The purpose of the activation function is to introduce non-linearity so that model is able to learn complex patterns. Different activation functions are discussed in later. There can be more than one hidden layer in a neural network. E.g., in the figure 4.1 there are three hidden layers named h_1 , h_2 and h_3 . The result from the hidden layers is passed to output layer.
3. **Output Layer** - This is the last layer and it is responsible for the output or result.

4.4.1 Activation Functions

These functions are one of the most important parts which help the model make decisions. These functions define whether the neuron would be activated or not by calculating the weighted sum of the input and transforming it to a different output. An activation function is non-linear to allow a neural network to learn patterns in the data and differentiable to allow backpropagation of the model's error to optimize the weights. They are used within the processing of the node. There are several different types of activation functions :-

1. **ReLU** - Rectified linear Units, if the input is greater than 0 then it returns the same value, otherwise it returns 0 if the input is less than 0.

$$R(z) = \begin{cases} z, & \text{if } z > 0. \\ 0, & \text{if } z \leq 0. \end{cases} \quad (4.1)$$

This function is commonly used in convolutional neural networks. It accelerates the convergence of gradient descent to find the global minimum. Its limitation is that the negative values are mapped to zero which hinders the model's ability to learn.

2. **Leaky ReLU** - This is a variant of ReLU, instead of returning 0 when the input is less than 0, this function returns a small value which is $input \times \alpha$.

$$R(z) = \begin{cases} z, & \text{if } z > 0. \\ \alpha z, & \text{if } z \leq 0. \end{cases} \quad (4.2)$$

This can be considered as an upgraded version of ReLU as it also considers the negative values. Its limitation is that the gradient for negative values is really small which increases the training time of the model.

3. **Sigmoid** - This function is also known as a squashing function as it takes a real number and outputs a number between 0 and 1.

$$S(x) = \frac{1}{1 + e^{-x}}$$

This function is commonly used where it is required to predict the probability as an output. Since probability is always in the range of 0 and 1, sigmoid is the correct choice. Its limitation is that it suffers from the vanishing gradient problem, i.e., the gradient becomes very small.

4. **TanH** - This function takes a real number and squashes the output between -1 and 1.

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

This function is commonly used in hidden layers because it is zero centered which means that the negative values will be mapped to strongly negative and the zero inputs will be mapped near zero. Its limitation is also that it suffers from the vanishing gradient problem.

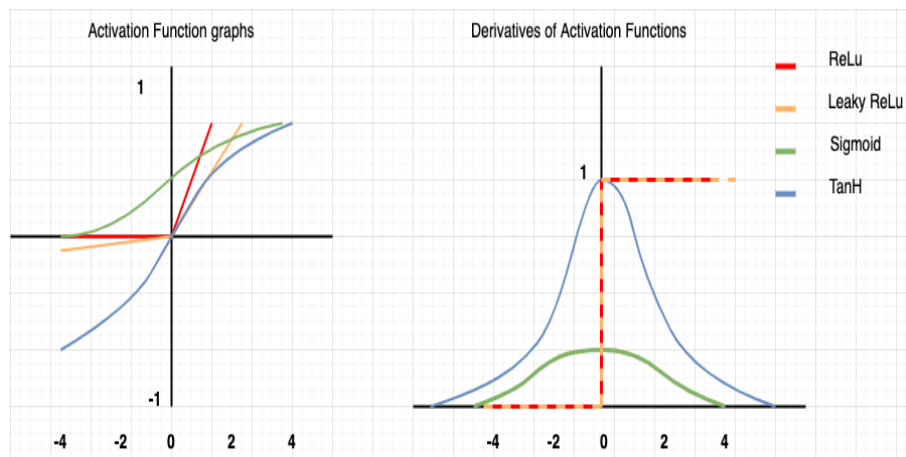


Figure 4.4: Graphs of activation function and their derivatives

4.4.2 Backpropagation

When a model makes a prediction it is evaluated against the truth value. The evaluation is based on a cost function. The objective of a neural network is to minimize the cost function. When a model makes an error in prediction it rectifies it error with a method called Backpropagation. It can be considered as one of the fundamental part of neural network. It trains a neural network through a method called chain rule. Forward pass is when network passes the information to next layer, while backpropagation performs a backward pass in which it adjusts the model weights and biases. In simple words, backpropagation aims to minimize the cost function by adjusting the weights and biases of the model. Backpropagation works by calculating the gradient of the cost function. The gradient is the partial derivative of the cost function which tells how much the output will change with respect to input.

1. Initially, the network get the input information from input node and weights are assigned randomly.
2. Information is processed and passed from input layer to hidden layer. Then hidden layer performs computation and gives results to output layer. The output layer gives the results according to the computation performed by hidden layers.
3. The error in the output is measured.
4. Goes back from output layer to previous layers while adjusting the weights. This process is repeated until the error is minimized.

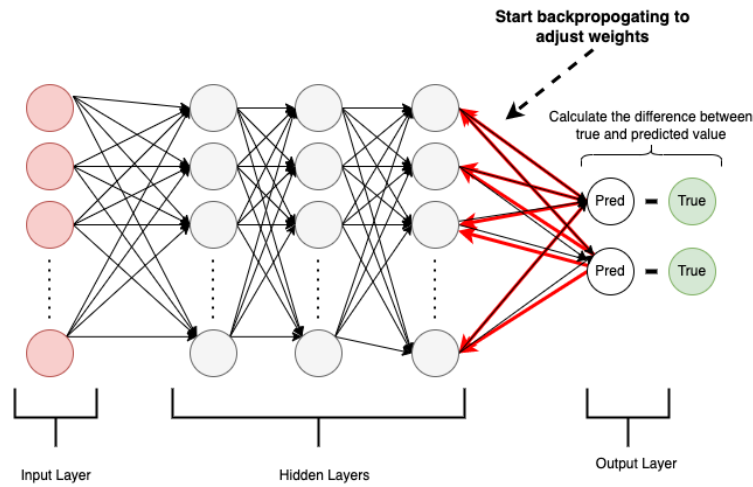


Figure 4.5: The figure shows how the network back propagates to adjust the weights. The red arrows shows the weight between output layer and hidden layer being adjusted.

4.4.3 Convolution Neural Network (CNN)

This type of neural network is mostly used in computer vision domain. The model is specialized in working with images because of its ability to capture features. For e.g., if the model is fed with lots of images of boy and girl it will learn to distinguish between both gender by itself. CNN is able to perform such computation with the help of convolution, pooling and parameter sharing.

Convolution Layer - The first layer of this model is known as the convolutional layer. This layer is responsible for extracting important features in the image by applying a filters on the image. The result obtained after applying filter on the image is known as "Feature Map".

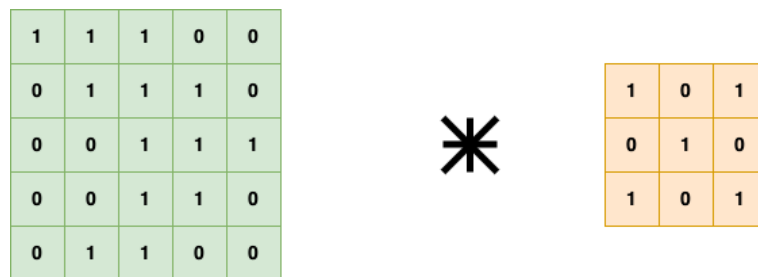


Figure 4.6: Consider the above figure as example where there is a 5 x 5 image matrix in green and a 3 x 3 filter in orange.

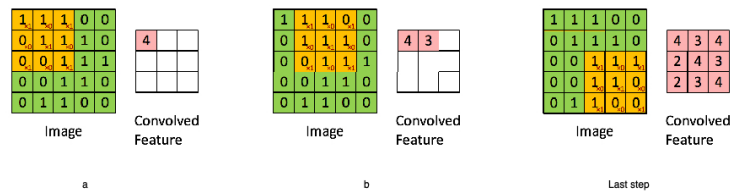


Figure 4.7: figure (a) describes how the filter is applied on the image matrix and sum of the products is stored as feature e.g., sum of the products in this case is 4 and this number is stored in convolved feature matrix. Similar step happens in figure (b) where the sum of the products is 3. Figure (c) shows the how the complete convolved feature maps look like

ReLU Layer - The rectified linear unit or ReLU is the next step after convolution layer. Feature map is put as input in ReLU which performs an element-wise operation on all the pixel by changing negative value to 0.

Pooling Layer - The next layer is the Pooling layer, this layer is responsible for reducing the dimensions or parameters when the image is too large. The advantage of using pooling layer is that it reduces training time and overfitting. The output from the ReLU is used as input in pooling layer which return pooled feature map. Pooling can be of different types :-

1. Max Pooling - it takes the largest element from the feature map.
2. Average Pooling - it takes the average value from the feature map.
3. Sum Pooling - it takes the sum of the elements from the feature map.

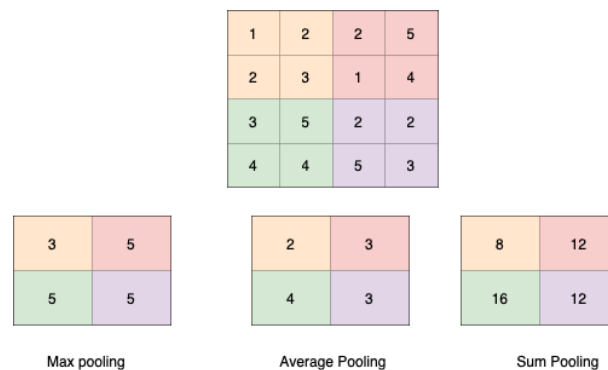


Figure 4.8: The figure explains 3 different types of pooling applied to 4 x 4 matrix.

Flattening - The next step is known as flattening, as the name suggests this process flattens the pooled feature map by converting the 2-dimensional pooled feature map to a single continuous linear vector.

Fully Connected Layer - This is the last layer of the CNN which is responsible for classification. It takes the flattened array as input and gives the output.

4.4.4 Recurrent Neural Network (LSTM)

This is another type of neural network but this model has ability to work with data that involves sequences. Other types of neural network are meant to work with the data where one data point is independent of another data point. But what if the new data point depends on the previous one?. This is where RNN plays an important role because it has a special component called 'memory' which helps it to store the information about previous inputs. In RNN information

cycles through a loop i.e., whenever it makes decision, it also considers information that it learned from the inputs it previously received or memory.

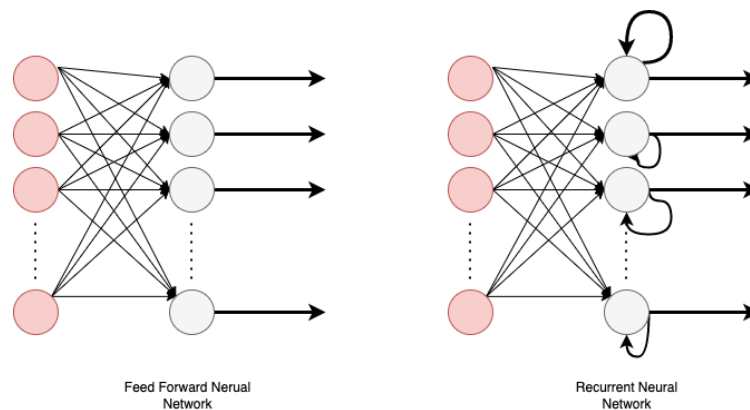


Figure 4.9: The figure explains how RNN uses feedback loop which helps the model to consider the memory and the current input to make the next decision.

4.4.5 Function of RNNs

This section will explain the working of the RNN model. The input data provided to RNN is in a form of sequence, this data is transformed into vectors. These vectors are then fed to RNN node. While doing computation RNN stores the information, this stored information is known as memory or hidden state. The hidden state is then taken into consideration while making decision for the next step. The next RNN cell then takes the hidden state and new information as input, these inputs are then passed to tanh function, the output of this function is then used as new hidden state.

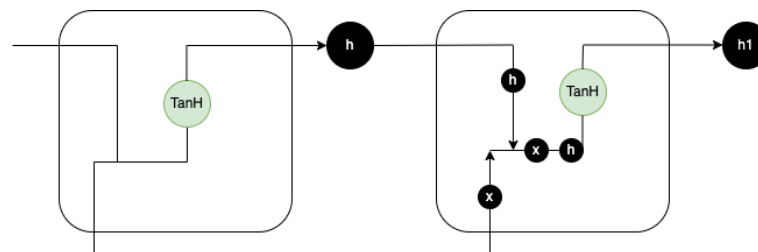


Figure 4.10: The figure explains how RNN works inside. The cell at the left side of the model passes the hidden state h , after computing it from the tanh function, to the next cell. The next cell takes the hidden state, h and new data, x as new input and passes it to tanh function to get new hidden state $h1$. This new hidden state will be used as hidden state for the next cell.

RNN model has two major problems short-term memory and Vanishing and Exploding gradient. **Short-Term memory** - The short term memory that is used to carry the information about previous steps, sometime becomes hard for the model to carry to next step if the sequence is long enough. This means that in case of long sequences the model might drop the information collected previously to be considered in current step.

Vanishing and Exploding Gradient - During training the RNN model the gradient sometimes can become too small or suddenly too large which results in low accuracy. Exploding gradient is caused when the weights are assigned high importance which results in gradient becoming too

large. Vanishing gradient occurs when the gradient decreases as it back propagates, which causes the model to stop learning.

Introducing Long Short-Term Memory (LSTM)

The Long Short-Term Memory model has the ability to forget or keep the information. The flow of information inside of LSTM is similar to RNN but what happens inside the cell is what that differentiates these two models. The two main components inside LSTM are cell state and gates. The cell state is sort of like a road that carries information from start to end of the sequence, we could think this as a memory for the model. This solves the short-term memory as the cell state carries information of whole sequence. The information in this cell state is changed by the gates. These gates in the model like neural network that adds or removes the information in cell state. These gates are able to learn which information to keep or discard during training of the model.

Different gates in LSTM

1. **Forget Gate** - This gate is responsible for keeping and discarding the information. The sigmoid function takes the information from the hidden state and new information as input and outputs a value between 0 and 1. If the value is 0 it means to forget the information and the value closer to 1 means keep the information.

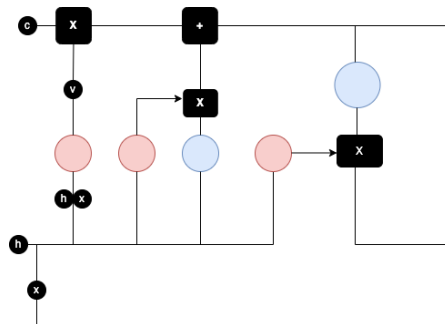


Figure 4.11: The figure above shows how the hidden state(h) and the new data (x) is used as input in the sigmoid function(shown as red circle).

2. **Input Gate** - This gate is responsible for updating the cell state. In this the hidden state and new data is used as input in two functions, first is the sigmoid function (similar to forget gate) and second is the tanh function that squeezes the output in the range -1 to 1. The result from both of these function is then multiplied. The sigmoid function is responsible for deciding which information to keep from the tanh.

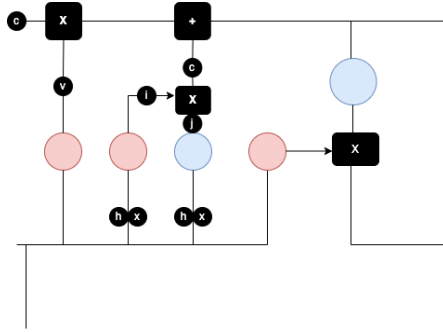


Figure 4.12: The figure above shows how the hidden state(h) and the new data (x) is used as input in the sigmoid function (shown as red circle) and the tanh function (shown as blue circle) the result from both of the function is then multiplied.

3. **Cell State** - This state is responsible for carrying the information from starting to the end of the sequence. The figure below explains how the cell state is changed. The result from the forget gate is multiplied then the result from the input gate is added. After these two steps it gives the new state.

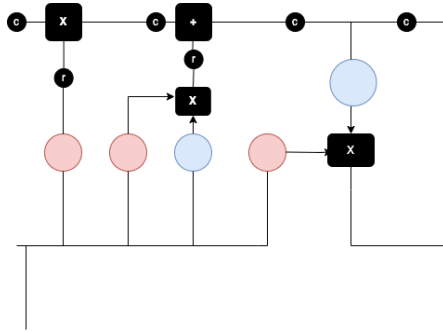


Figure 4.13: The figure above shows how the cell state(c) is updated. First, the cell state is multiplied by the output(r) of the forget gate and then the result(r) of the input gate is added to the cell state.

4. **Output Gate** - This state is responsible for the giving a new hidden state. The new modified cell state from forget and input gate is passed to tanh function. The current data and previous hidden state is passed to sigmoid function. The result from both of the function is then multiplied which becomes the new hidden state.

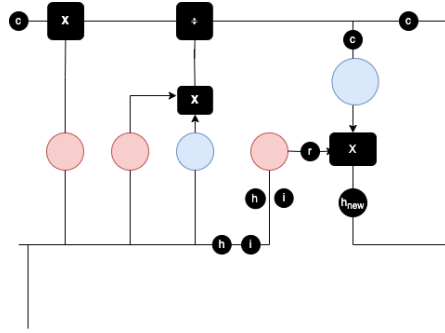


Figure 4.14: The figure above shows how the hidden state (h) and the new data (x) is used as input in the sigmoid function (shown as red circle). The cell state (c) is passed through the tanh function. The result from both the function is then multiplied. The product then becomes the new hidden state (h_{new})

4.4.6 CNN + LSTM model

This section will explain the design of the CNN + LSTM model used for the project. The blue blocks in the figure 4.7 is used as convolutional layer. The main role of these layers is to extract features as CNN is capable of extracting features that are scaling and rotation invariant. It can also extract spatial features such as angles and distance between the coordinates of the keypoints in the frame.

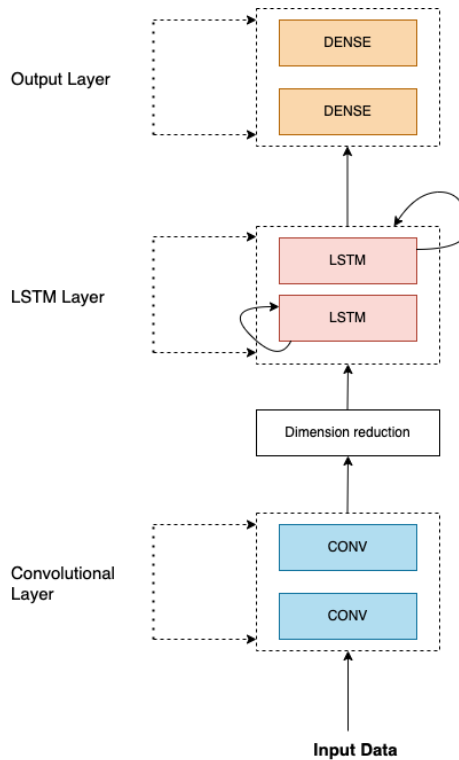


Figure 4.15: CNN + LSTM model architecture

The red blocks in the model represents the LSTM part of the model. LSTM is commonly used for predicting time-series data and since the data used by the model is a sequence of frames,

LSTM helps the model to learn the changes by identifying the changes in the features of the input data extracted by LSTM layers. The final layer or output layer is a fully connected layer that takes the data provided by the LSTM as input and outputs the type of motion predicted. In short, the CNN part of the model extracts the features from the input tensor of poses and LSTM part of the model recognizes the changes in the features of the data in each frame.

5 | Methodology

This chapter will explain the steps taken in the project to meet the aims explained in chapter 1.

5.1 Given dataset

To prepare the dataset for different types of motions, a zipped folder was provided. The zipped folder contained the following items:-

1. csm data of 30 different characters - Each folder contained numerous csm files. These files contain marker-position data of the character. It contains the information about, when the data was taken, how it was taken, number of frames in the file, capture rate of the motion, the joint movement of the character in form of coordinates, naming convention of each joint of the character. Each csm file represents a specific type of motion.
2. Key.pdf - This pdf file contained the information about all the characters in the folder e.g., name, age, gender etc.
3. alt_naming_convention_CSM.pdf - This pdf file contained the naming convention used for the joints. E.g., LFHD meant LFrontHead, STRN meant Chest etc.
4. Problems with the data.pdf - This pdf file contains the information about any problem related to the data.

Along with this zipped folder, a scripts folder was provided. This folder contained the scripts which were to be used in Autodesk 3ds Max for creating animations.

5.2 Overview of the framework

The animations of different types of motions were created with the help of the CSM files and scripts using 3dsMax. The animation required a Mixamo character which was applied to animation model. After a motion is animated, video of that particular animation was recorded. The videos were named in the same format as that of CSM files. The videos in the database were recorded in 60 frames per seconds and were about 11 to 13 seconds long.

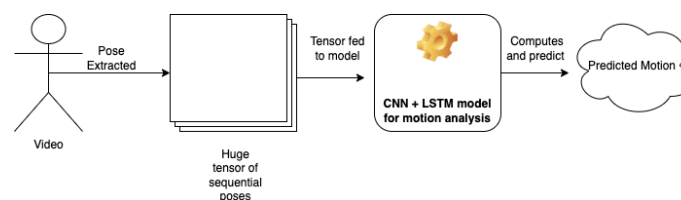


Figure 5.1: The figure shows a summarized diagram how the flow of project

After all the videos were recorded, the next step was pose extraction. The process of pose extraction was done using openPose. The poses were estimated and contained in JSON file, the

file had 25 sets of x and y coordinates belonging to each person in the video, for our project there was just one person in the video. OpenPose extracts the pose on each frame of the video, so if a video is 11 seconds long and there are 60 frames per seconds, then there will be, $11 \times 60 = 660$ poses in form of json files.

Once the pose extraction process is completed then the x and y coordinates from all the json files of a particular video are read and stack on each other to form a large tensor of poses and saved in *.npy* format. The same step is repeated for all the other videos. Then all the *.npy* files are concatenated to form a single large tensor for training the model. The labels for the training data is produced in a similar way but with one-hot encoding (further explained in section 6.4). After the training data was prepared, it is then used as an input along with the labels in the CNN-LSTM model. The model accepts and classifies the data framewise.

The next couple of section will explain each of the step (described above) in detail.

5.3 Animating motions in 3ds Max

This section will explain how the CSM files and scripts were used to make the animation with the help of 3ds Max. But before jumping on the progress of how to make the animation lets first learn a bit about 3ds max.

5.3.1 Autodesk 3dsMax

3ds Max is an Autodesk software used for creating high-quality 3D animations. This software also has modelling abilities and provides many plugin but can only be used on Microsoft windows platform. It is commonly used by TV commercial studios, game developers or movie creators. 3ds Max software was used to create the animation because the CSM files for different motions were given. The CSM file stores motion-capture data which is data of motion from the character performing a particular action. The files when imported, only takes the marker position data. 3ds Max uses the CSM file to extract limb rotation data to position the character.

5.3.2 Creating animations

This section will explain how the animations were made. Since 3ds max can only be used on Microsoft windows platform, a windows computer was used (provide specs) The process to animate the character were as follows: -

1. First, a biped model was made. A biped model is a figure that looks like a human. This model have properties which allows it to be ready for animation. Its appearance is similar to a human skeleton

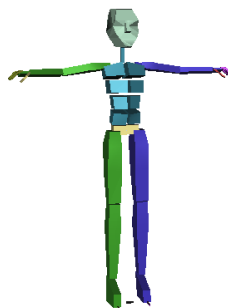


Figure 5.2: Biped Model

2. When the biped model is made, under the motion capture file, we upload one of the CSM files. Once the CSM file is loaded in the biped the model will change its action to T-figure. Successfull loading of CSM file can be checked by playing the animation.
3. This animation is then stored as a .bip file.
4. After the animation file is save, open the character downloaded from the Mixamo in 3ds Max. The Character will be in T-position.

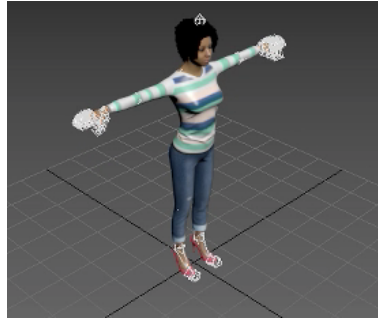


Figure 5.3: Biped Model

5. Click on the RunScript option and run the AutoBiped.ms file provided in the zipped file. This script changes applies the skin to the skeleton for the animation to work properly.
6. Once it is done, load the animation file (.bip) that was saved in step 3 and run the animation.

Once these steps are completed the character downloaded from Mixamo plays the motion related to the CSM file. These steps were repeated for all the CSM files. Once the animations were done, it was then recorded in 60 frames per seconds(fps). Each of the videos were about 11 – 13 seconds long.

5.4 Pose Extraction

This section will discuss how the poses were extracted from animation videos. As mentioned previously OpenPose was considered to extract the poses from videos. OpenPose was used in Google colab, the colab file provided the commands necessary to install and build openpose. The main idea of the pose extraction was to extract poses from each frame of the video, so if there was a video of 11 seconds in 60 fps then there will be $11 \times 60 = 780$ different poses. Each of the poses had 25 keypoints and were stored in a json file (i.e., 780 json files). Each file had poses in the format x, y, c (c is the confidence score).



Figure 5.4: The figure shows the location of all the keypoints that were extracted during pose estimation.

To extract the poses from all the video, all the videos were uploaded to the colab file and then the pose extraction process was initiated. All the poses were saved in the file that had the same name as that of the video and were uploaded to drive to maintain a backup as colab deletes all the uploaded file after 24 hours. After pose extraction method comes the process of combining poses.

5.5 Combining poses and making labels

This section will explain how the poses of the all the videos were combined along with making the labels for the video.

The poses of each video were combined one by one because the videos were of different length so there was a need to standardize the number of poses and also to ensure that each file is being read properly. The steps below explains how the poses were combined to form a large tensor of poses.

1. Each of the folder contained numerous json file containing poses. Each of the file in the json folder was read, if the video was of 11 seconds then 660 frames were read and if the video was of 13 seconds then 780 frames were read.
2. In each json file, all the poses under the pose pose_keypoints_2d section were read. All the x and y coordinates were taken separately with indexing technique and stacked depth-wise to return a 3d array.
3. This processes was repeated for each json file and each set of x and y coordinates were stacked on top of each other to form a large array.
4. At the end, for a 11 seconds long video the shape of the array was (660, 25,2). This array was saved in the .npz format, it is format in which numpy array are saved.
5. When all the poses of videos were combined then all the .npz files were read and combined to form a single file that would contain all the different types of poses. To achieve this, All the .npz files were read and reshaped according to (total_length_of_video, number_of_frame, 25, 2). After reshaping all the .npz files, it was vertically concatenated on top of each other and the order in which they were stacked was noted to make the appropriate labels.

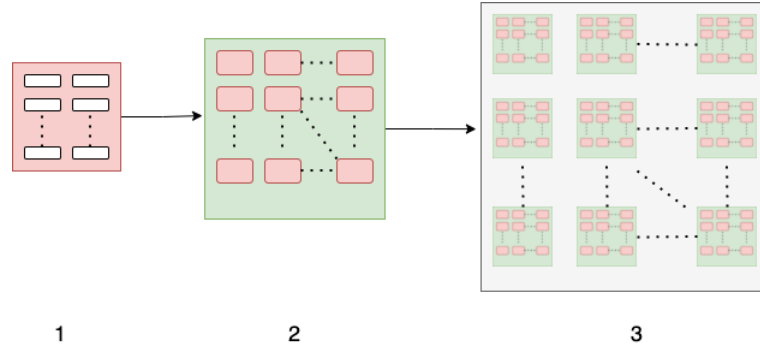


Figure 5.5: In the figure, the red box marked as 1 shows the 25 pairs of x and y coordinates with shape(25,2). The green colored box marked as 2 shows numerous red box which represents 60 frames of x and y coordinates. The final grey box marked as 3 represents shape total_length_of_video, number_of_frame, 25, 2)

After the process of combining all the poses was done then the labels were made. The videos had three different labels i.e., knocking, lifting, throwing and walking. To make the labels One-hot encoding technique was used, it is a process in which categorical variables such as 1, 2, 3 and 4 are converted into a form that could be provided to models to improve the prediction. The label knocking was converted to $[1., 0., 0., 0.]$, lifting was converted to $[0., 1., 0., 0.]$, throwing was converted to $[0., 0., 1., 0.]$ and walking was converted to $[0., 0., 0., 1.]$. The labels were vertically stacked so that they would label each frame in the pose tensor. To achieve this the shape of the labels was total_length_of_video, number_of_frames, 4 (the last element is 4 as the length of each encoded label is 4).

5.6 Structure of model

This section will explain how the model was implemented. The model used for motion analysis is a combination of CNN and LSTM. The CNN part of the model is used for extracting features from tensor and LSTM part is used for analysing how the features are changing over the course of time. The model uses time-distributed CNN layer to extract different features from coordinates present in the training data (explained in previous section). The LSTM layer analyses the changes after each frame. The model accepts the input of shape $60 \times 25 \times 2$, this means 60 sequential frames with 25 sets of coordinates named X and Y.

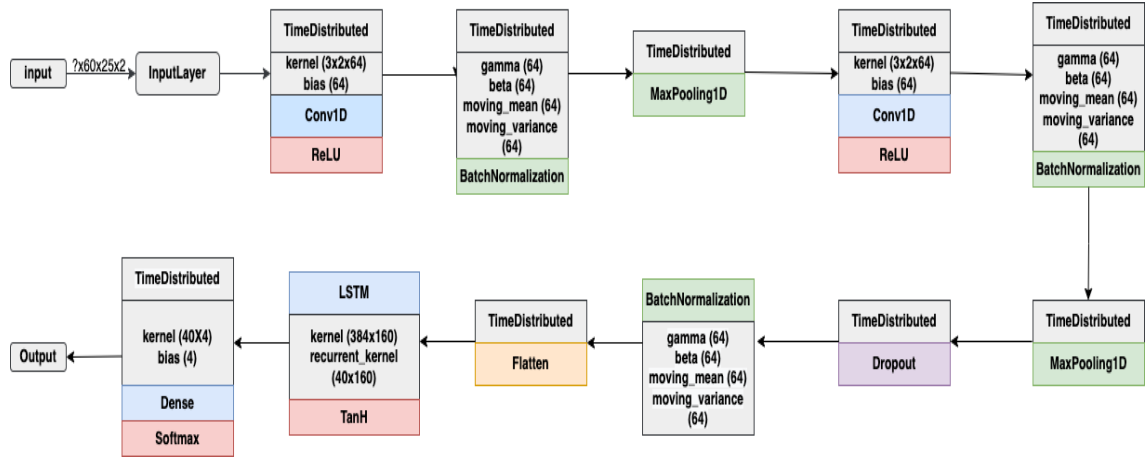


Figure 5.6: Model flow

The model started by first creating an object of sequential class. Sequential was used as it has the ability to stack different layers to a model. Time-distributed CNN layer with 64 filters of size 3 with activation layer ReLU was applied to each frame of the tensor, Time distributed was used as it is very useful when working with time series data or frames of a video because it applies same set of weights to each of the frame, this means that it will apply same set of weights for 60 frames. The conv layer in time distributed extracts features from the tensor. Then time distributed batch normalization was applied to make the neural network more stable and faster, it basically standardizes (re-scaling data to have mean of 0 and standard deviation of 1) the inputs of a layer for each mini-batch. After Batch normalization, time distributed max pooling was applied to reduce the dimensions of feature map. After pooling, the whole process of convolution, batch normalization and pooling was done again in time distributed wrapping. This was followed by time distributed dropout layer as it prevents overfitting the data. Overfitting is problem that occurs when model has a good performance in training but poor performance when it comes to testing of the data. To avoid this issue, batch normalization drops the output of the layer. The dropout probability was chosen as 0.5. The result from these layers was then again applied to batch normalization followed by time distributed flatten, this was done to convert the result to 1-D data. The flattened data was then passed to LSTM layer with 40 units with unit forget bias as 0.4. The final or output layer of the model was time distributed layer that had 4 outputs to represent 4 different types motion (walking, knocking, lifting, throwing) and softmax as its activation layer. The softmax function takes input and convert it to probability distribution. This function is commonly used in multi-class classification tasks.

6 | Evaluation

This section will discuss the results after training the model on the poses. The model was compiled with categorical cross-entropy as loss function because it is commonly used for multi-class classification tasks. Adam optimizer with learning rate of 0.0001 rate was chosen as the optimizer for the model to control the learning rate of the model. The model was trained for 200 epochs on Macbook pro with 2 GHz Quad-core Intel core i5 processor, 16 GB 3733 Mhz memory and Intel Iris Plus Graphics 1536 MB graphics.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 60, 25, 64)	448
time_distributed_1 (TimeDist	(None, 60, 25, 64)	256
time_distributed_2 (TimeDist	(None, 60, 12, 64)	0
time_distributed_3 (TimeDist	(None, 60, 12, 64)	12352
time_distributed_4 (TimeDist	(None, 60, 12, 64)	256
time_distributed_5 (TimeDist	(None, 60, 6, 64)	0
time_distributed_6 (TimeDist	(None, 60, 6, 64)	0
batch_normalization_2 (Batch	(None, 60, 6, 64)	256
time_distributed_7 (TimeDist	(None, 60, 384)	0
lstm (LSTM)	(None, 60, 40)	68000
time_distributed_8 (TimeDist	(None, 60, 4)	164
Total params: 81,732		
Trainable params: 81,348		
Non-trainable params: 384		
(None, 60, 4)		

Figure 6.1: CNN + LSTM model summary

The motive of this task was to recognize the type of motion with maximum accuracy. First, the coordinates of joints are extracted from the motion videos into a JSON file, each file is then read and concatenated to form one large tensor of training data, after this the CNN-LSTM model is then trained on this data along with the validation data. The model was trained in two different ways. First, the model was trained on mixed videos on male and females, then evaluated on different videos. Second, the model was trained on just male videos and then evaluated on female videos. The sections below discusses both the approaches in terms of accuracy and loss of the model during the training and validation phase followed by the result showing the confusion matrix, F1 score.

6.1 Metrics used

Before jumping to results of the model, this section will explain the metrics used to evaluate the performance of the model.

6.1.1 Loss

The loss of a model indicates how badly the model's prediction was on a test case. Higher loss means that the model is making bad predictions and lower loss means that the model is making good prediction. The model uses categorical cross-entropy as its loss function because this loss is generally used for multi-class classification tasks. The mathematical formula for this loss is,

$$Loss = \sum_{n=i}^{outputsize} y_i \cdot \log \hat{y}_i$$

\hat{y}_i is the i-th value that the model predicted, y_i is that particular target value and the output size is the number of values that the model had to output. The main objective of the model is to achieve high accuracy and low loss.

6.1.2 Accuracy

While training the model we will calculate the loss and accuracy of training and validation set. Accuracy is one of the popular metric to evaluate the performance of the model. The formula for calculating the accuracy of a model is

$$Accuracy = \frac{NumberofCorrectpredictions}{Totalnumberofpredictions}$$

The accuracy of the model was measured with the help of **sklearn.metrics** module, this module has a function named **accuracy_score** which takes *true_results* and *predicted_results* as parameters and returns the accuracy score. It returns the value between 0-1 which can be multiplied by 100 to show the accuracy percentage.

6.1.3 precision

This metric tell how often the model is making correct positive prediction. Precision is calculated as the ratio between true positives to the total number of true positives and false positives. Precision tells how accurate the model is when it predicts a test case as positive.

$$Precision = \frac{TP}{TP + FP}$$

The precision score of the model was measured with the help of **sklearn.metrics** module, this module has a function named **precision_score** which takes *true_results* and *predicted_results* as parameters and returns an array of precision score. This method returns an array that shows the precision score for each class.

6.1.4 recall

This metric tell about the model's capability to detect positive samples. Recall only deals with the cases that were classified as positive i.e., value returned by the recall tell the number of cases that were classified as positive.

$$Recall = \frac{TP}{TP + FN}$$

The recall score of the model was measured with the help of **sklearn.metrics** module, this module has a function named **recall_score** which takes *true_results* and *predicted_results* as parameters and returns an array of recall score. This method returns an array that shows the recall score for each class.

6.1.5 F1 score

This metrics provides a way to evaluate model's precision and accuracy. High F1 score denotes less false positive and false negatives, this means that model with F1 score closer to 1 is considered as a good model than a model which has a score closer to 0.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

The F1 score of the model was measured with the help of **sklearn.metrics** module, this module has a function named **f1_score** which takes *true_results* and *predicted_results* as parameters and returns an array of F1 score. This method returns an array that shows the f1 score for each class.

6.1.6 Confusion matrix

As the name suggest the result of this method is a matrix that shows the results of all the predictions in a summarized manner. Each row in the matrix symbolizes a predicted class and each column in the matrix symbolizes an actual class.

	YES EVENT	NO EVENT
YES EVENT	TRUE POSITIVE	FALSE POSITIVE
NO EVENT	FALSE NEGATIVE	TRUE NEGATIVE

Figure 6.2: The figure above shows format of the confusion matrix.

The figure above shows the binary classification between two classes. The structure of the matrix is as follows

1. **True Positive, TP** - Represents the number of correct predictions.
2. **False Positive, FP** - Represents the number of incorrect predictions.

3. **True Negative, TN** - Represents the number of correct predictions of "NO EVENT".
4. **False Negative, TN** - Represents the number of incorrect predictions for "NO EVENT".

The confusion matrix of the model was calculated with the help of `sklearn.metrics` module, this module has a function named `confusion_matrix` which takes `true_results` and `predicted_results` as parameters and returns an multi-dimensional array known as confusion matrix. The multi-dimensional array returned by this method shows prediction for each of the class. Confusion matrix shows how confused the model is when it makes prediction. This metric is useful as it can be used to calculate most of the other metrics discussed above.

1. Accuracy = $TP + TN / TP + TN + FP + FN$
2. Precision = $TP / TP + FP$
3. Recall = $TP / TP + FN$

6.2 Training - Approach 1

In this approach the model was trained on mixed videos of male and female. The training data had a shape of $502 \times 60 \times 25 \times 2$, this denotes that model was trained on total for 502 seconds with 60 fps. The validation data had a shape of $242 \times 60 \times 25 \times 2$.

6.2.1 Accuracy And Loss

Figure 6.3 shows the comparison in accuracy between training and validation phase of the model. Initially the curve increases rapidly which is a positive sign as it shows that the model is learning to classify the videos. There are no steep slopes after almost 75 epochs which shows that the model from there started to stabilize. In the end, the model reached the accuracy of almost 95% accuracy and 96% accuracy during training and validation.

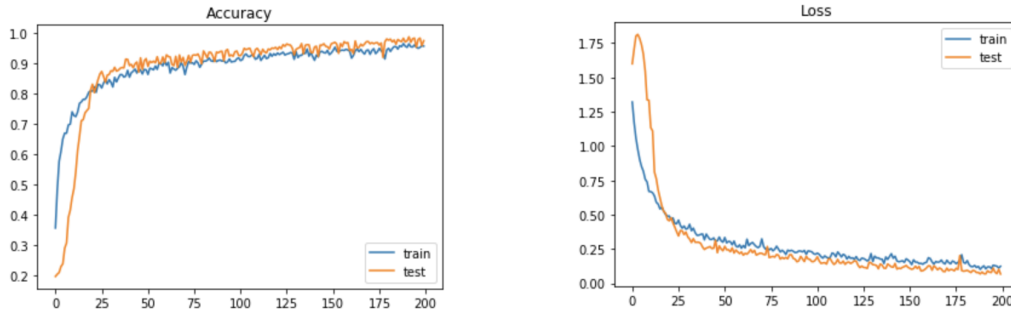


Figure 6.3: Accuracy and Loss of model trained on male and tested on female

Figure 6.3 shows the difference in the loss of the model between training and validation phase. Initially, the training and validation loss started at high value, which was then followed by the steep slope. The loss in the graph started to stabilize after almost 75 epochs (similar to accuracy graph in figure 7.1). Towards the end, training and validation of the model had loss less than 0.25.

6.2.2 Results

After training and validating the model on training and validation tensor, the model was tested to classify 480 frames i.e., 8 videos (as each video was recorded in 60 fps so $\frac{480}{60} = 8$). The result can be seen with the help of confusion matrix in the figure 6.4

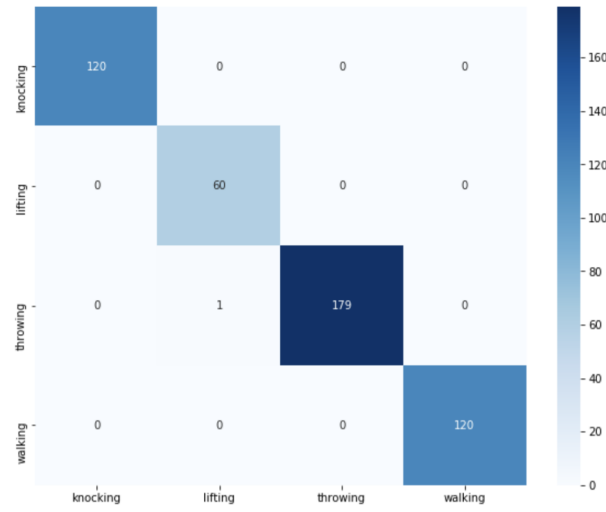


Figure 6.4: Confusion matrix after testing the model on 8 different videos

The figure 6.4 shows that the model performed well as it was able to classify almost all the frames of the video. Only 1 frame was incorrectly classified as lifting but it was throwing. The model had accuracy of 99.79% at the testing stage. The table in the figure 6.5 shows the F1, precision and recall score of model according to the classes.

	Knocking	Lifting	Throwing	Walking
F1	1	0.9917	0.9972	1
Precision	1	0.9836	1	1
Recall	1	1	0.9944	1

Figure 6.5

6.2.3 Summary of Approach - 1

The CNN-LSTM model was trained on videos of both male and female videos for 200 epochs, the training set had shape $540 \times 60 \times 25 \times 2$ and validation set had shape of $176 \times 60 \times 25 \times 2$. After training and validation the model had accuracy of 97.4% on training data and 97.5% on validation data. The model was then tested on 8 videos of both male and females. Out of 480 frames, only 1 frame was incorrectly classified which gives the accuracy of 99.7% on testing case. F1, recall and precision scores are shown in the figure 6.5. The figure below shows the overall classification of the model.

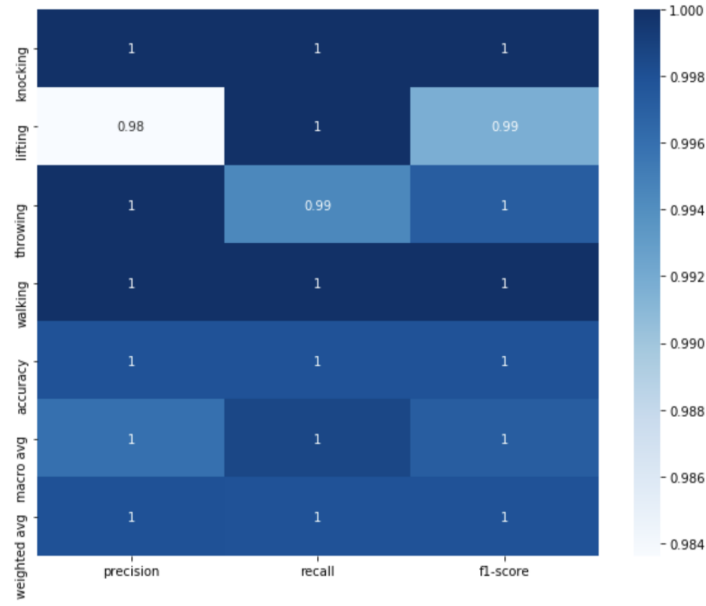


Figure 6.6

6.3 Training - Approach 2

In this approach the model was trained on the male videos and then evaluated on the basis of female videos. This was done to identify how the model will classify motion on the basis of gender. The training data had shape, $220 \times 60 \times 25 \times 2$, this denotes that model was trained on total for 220 seconds with 60 fps. The validation data had shape, $144 \times 60 \times 25 \times 2$

6.3.1 Accuracy And Loss

Figure 6.7 shows the comparison in accuracy between training and validation. During the training phase of the model the model was performing well which is depicted by the steep slope in the accuracy graph. The accuracy of the model during the validation reached to almost 95%. But, in between the validation phase of the model there is a slight dip between 15 to 40 epochs which shows that model was not able to perform well on the female poses data as towards the model had accuracy of nearly 80%.

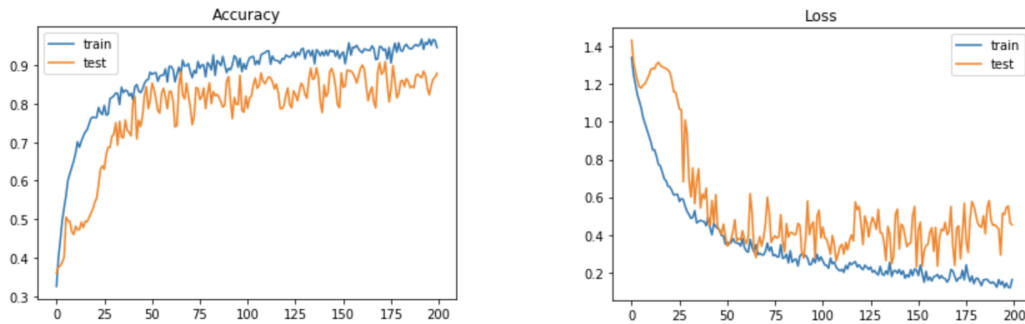


Figure 6.7: Accuracy and Loss of model trained on male and tested on female

Figure 6.7 shows the difference in the loss of the model between training and validation. During the training phase the model shows no problem as towards the end it had the loss of almost less than 0.1. But during the validation the loss slightly increases between 15 to 40 epochs, this fact can be verified by looking at the accuracy graphs as the accuracy of the model shows a dip between the same interval.

6.3.2 Results

After training and validation the model, the model was tested to classify 420 frames i.e., 7 videos (as each video was recorded in 60 fps so $\frac{420}{60} = 7$). The video originally had 2 knocking, 1 lifting, 2 throwing and 2 walking videos. The result can be seen with the help of confusion matrix in the figure 6.4

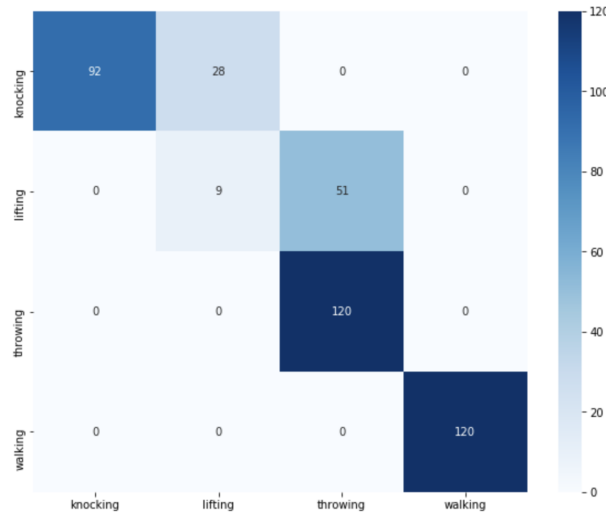


Figure 6.8

The figure 6.9 shows that the model was able to classify the frames of throwing and walking without any problem as it correctly predicted 120 frames of both throwing and walking. But the model was confused between knocking and lifting videos as 28 frames of knocking were classified as lifting and 51 frames of lifting were classified as throwing.

	Knocking	Lifting	Throwing	Walking
F1	0.867	0.185	0.824	1.
Precision	1.	0.243	0.701	1.
Recall	0.766	0.15	1.	

Figure 6.9

6.4 Comparison with Baseline Model

The baseline model selected for this project is a CNN model. This model was selected because the project deals with tensor and the model that is commonly used for extracting features from tensors is the CNN model. The specification of the baseline model was kept same to the CNN part of the hybrid model. The model was trained and validated on the same data. The results are below.

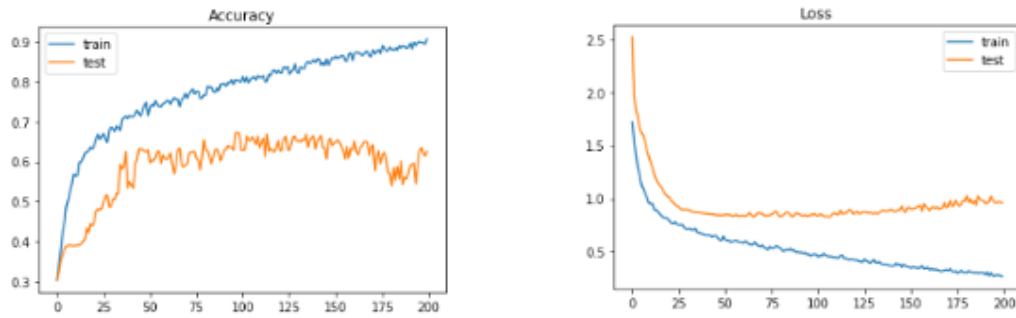


Figure 6.10: The figure above shows the accuracy and loss of the baseline model during training and validation. The model had the accuracy of almost 60% during validation whereas the CNN + LSTM model had the accuracy of 95%. Similar case can be seen in the lose graph where the loss of the baseline model does not decrease any further. after 25 epochs.

6.5 Testing on Real-life data

This experiment was conducted to analyse how them model that is trained on virtual character motion videos will behave with real person's video. To verify this objective, the model used in approach 1 was taken as it was trained on both male and female videos and had the best metrics as compared to the model used in approach 1. To conduct this experiment 4 people were selected. They were asked to do the motion similar to the ones in the dataset. The videos were recorded in the 60 fps and were almost 5 seconds long.



Figure 6.11: The figure above shows the picture of a subject doing a knocking motion. The colorful line shows the process of pose extraction done on the video.

The results of the test can be seen in the confusion matrix in 6.11, total of 720 frames were classified. It can be clealy seen that the model is

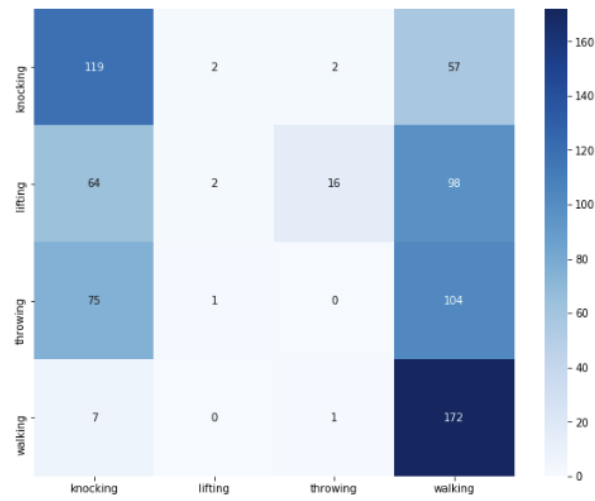


Figure 6.12: The figure shows the confusion matrix when the model was tested on real-data

Even though the model did not perform well but this could still be considered as a positive point as all the walking frames were classified correctly and more than half of the knocking frames were classified correctly. The experiment conducted in this section can be seen as proof of how machine learning can close the gap between virtual and reality.

7 | Conclusion

7.1 Summary

We first made the dataset required for the model in 3ds Max with the help of the csm files and scripts which were initially provided. The dataset contained the four different types of motion which were knocking, lifting, throwing and walking. The motions were of different characters. The animations were recorded in 60 fps and were on an average 11–13 seconds long. We then extracted poses from each of the videos with the help of pose extraction library called OpenPose. The poses were extracted and contained in json file in form of 25 sets of keypoints. The number of json file created were equal to the total number of the frames in the video. Each of the json file was then read and stacked to form a large tensor of poses. We then implemented a model that had the features of both CNN and LSTM. The CNN part of the model was used for feature extraction and LSTM part for sequence prediction. The model takes a frame as an input and classifies each frame. The model had four different outputs which represents the four different types of videos. The model was then trained on two different approaches. First, the model was trained and validated on both male and female videos for 200 epochs. In this approach the model had the accuracy of $\approx 95\%$ during testing and $\approx 95\%$ during validation. Second, the model was trained on male videos and validated on female videos. In this approach the model behaved differently as during training the model had the accuracy of $\approx 95\%$ but during validation it had the accuracy of $\approx 80\%$. Then an experiment was conducted to analyse how the model that is trained on virtual data will behave with real world data.

7.2 Summary of contributions

This project has two major contribution

- This project provides a overview of the system about how pose extraction and machine learning could be used together to analyse different types of human motion.
- This project provides a way of how a virtual data (animations) could be beneficial for the machine learning model in real-life scenarios.

7.3 Limitations

1. **Limited dataset.** This project was mostly dependant on the dataset provided but unfortunately few of the csm file in the provided folder had corrupted data which resulted in less animation video for the data. This effect of this could be seen in the section 6.3 where the model was trained on male data and tested on female data, the animation videos of female character were less in compared to male one which resulted in less testing on female character videos. Getting more csm files would have helped in creating more animation videos.
2. **Realistic characters.** The project focused on getting the most realistic character as the realistic character would helped the project to get trained on characters that had features

closer to real world. The metahuman was considered as an option but at the current time could not be used in 3ds Max. If the model was trained on the characters from metahumans then maybe we could have got a better results from pose extraction methods.

3. **Enhancing pose extraction** - In the pose extraction method the project currently considers only the main body part but not the fingers. Few of the motion like knocking, lifting and throwing are similar, the only way to differentiate between them is to extracting the positions of fingers. E.g., while knocking the person usually knocking with back side of their hand with two fingers coming forth and while pressing the button of the lift, the person takes out the index finger to press the button. But, the csm file provided contains no information about the position of the character's fingers.



Figure 7.1: The picture in the left is knocking and the picture in the right is pressing the button of lift. It can be seen in the picture that the character is using his full hand in both videos

7.4 Future work

This project can be improved in several ways:-

1. **Training the model on more realistic data.** The model was trained on the animations of Mixamo's character. In future if metahumans becomes compatible with 3ds Max then the model could be trained on the metahuman videos, this could result in improvement as the metahumans represent more realistic human features which could help in improving pose extraction.
2. **Increasing the size of dataset.** Increasing the size of dataset can help the model to get more data to train and tested which could further help in testing the model on real-life data. As discussed in the chapter 2, using gaming engines like Unreal Engine and simulators like Adultsim, the size of the dataset could increase which can help the model to diverse features.
3. **Adding some randomness to dataset.** Currently all the videos in the dataset are recorded from front video but in future the dataset can have different videos where the animations are recorded from different angles, changing the color of background, changing the features of characters such as clothing, skin color, hair etc. This could increase the quality of the dataset and could be beneficial as the model will be able to get the information about the whole body of the person. Also, if the project gets more csm file, then more animation from different angles can be recorded which could also increase the size of the database.
4. **Improving animations.** As discussed in the section 7.3, the animation does not take the fingers of the characters into account. If the fingers were also taken into account then there be some improvement between the knocking, lifting and throwing video.

A | Appendices

A.1 Ethics information: Ethics consent form

**School of Computing Science
University of Glasgow**

Ethics checklist form for 3rd/4th/5th year, and taught MSc projects

This form is only applicable for projects that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please contact the Chair of the School of Computing Science Ethics Committee (matthew.chalmers@glasgow.ac.uk) for advice.

If your evaluation does comply with all the points below, please sign this form and submit it with your project.

-
1. Participants were not exposed to any risks greater than those encountered in their normal working life.

Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

2. The experimental materials were paper-based, or comprised software running on standard hardware. *Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, laptops, iPads, mobile phones and common hand-held devices is considered non-standard.*

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.

The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

5. No information about the evaluation or materials was intentionally withheld from the participants.
Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.
6. No participant was under the age of 16.
Parental consent is required for participants under the age of 16.
7. No participant has an impairment that may limit their understanding or communication.
Additional consent is required for participants with impairments.
8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.
9. All participants were informed that they could withdraw at any time.
All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.
10. All participants have been informed of my contact details.
All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.
11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. In cases where remote participants may withdraw from the experiment early and it is not possible to debrief them, the fact that doing so will result in their not being debriefed should be mentioned in the introductory text.
12. All the data collected from the participants is stored in an anonymous form.
All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Project title _____ Human Motion Analysis _____

Student's Name _____ Kshitiz Bisht _____

Student Number _____ 2487508b _____

Student's Signature _____ Kshitiz _____

Supervisor's Signature _____  _____

Date _____ 29/03/2022 _____

A.2 Ethics information: Introduction and Debrief scripts

Ethics information: Introduction script

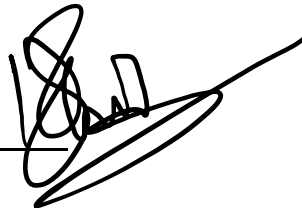
Human Motion Analysis
2021/22
Kshitiz Bisht

The aim of this experiment is to analyse performance of machine learning model that is trained on virtual data with real data. Making a model that can analyse human motion can be extremely useful as it has many applications such as diagnosing Parkinson's disease, surveillance. But making a model like this requires dataset for training and testing purposes. These datasets are limited and often requires consent of the individual. To tackle this problem the model could be trained on synthetic data, in simple terms training the model on videos of virtual characters. These videos can be made in gaming engines such as Unreal Engine. But ultimately these models will be used in real world, which is why we need to run an experiment where we will check the performance of the model on real world data. The model I have right now can analyse 4 different types of motions. You must perform actions such as knocking the door, throwing a ball, pressing the button of the lift and walking. I will make a video of everyone doing these actions one by one. Each video would be of 5-10 seconds long. Your videos will not be available for public use unless you give your explicit consent. Your consent for public availability will be asked in debriefing part. Please ask questions if you need to and please let me know when you are finished. Please remember that it is the model, not you, that is being evaluated. You are welcome to withdraw from the experiment at any time. Do you agree to taking part in this evaluation? If yes, then please sign in the space given below.

C. Knox



C. Montuth



Ethics information: debriefing scripts

**Human Motion Analysis
2021/22
Kshitiz Bisht**

The aim of this experiment is to analyse performance of machine learning model that is trained on virtual data with real data. I just wanted to record 4 videos of each participant doing different motions such as knocking the door, throwing a ball, pressing the button of the lift and walking. These videos will now be processed for pose extraction method and then will be used to analyse the performance of the model.

Do you have any comments or questions about the experiment? Please take a note of my email address (2487508b@student.gla.ac.uk), and please let us know if you have any further questions about this experiment. Thank you for your help.

If you want to give your consent to make your videos publicly available, please give your names to me.

7 | Bibliography

- M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Sequential deep learning for human action recognition. In *HBU*, 2011.
- Z. Cao, T. Simon, S. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1611.08050, 2016. URL <http://arxiv.org/abs/1611.08050>.
- R. Chen, M. Lu, T. Chen, D. Williamson, and F. Mahmood. Synthetic data in machine learning for medicine and healthcare. pages 1–5, 06 2021.
- J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014. URL <http://arxiv.org/abs/1411.4389>.
- H. Fang, S. Xie, and C. Lu. RMPE: regional multi-person pose estimation. 2016. URL <http://arxiv.org/abs/1612.00137>.
- H. Gammulle, S. Denman, S. Sridharan, and C. Fookes. Two stream LSTM: A deep fusion framework for human action recognition. *CoRR*, abs/1704.01194, 2017. URL <http://arxiv.org/abs/1704.01194>.
- Z. Geng, K. Sun, B. Xiao, Z. Zhang, and J. Wang. Bottom-up human pose estimation via disentangled keypoint regression. *CoRR*, abs/2104.02300, 2021. URL <https://arxiv.org/abs/2104.02300>.
- H. Hwang, C. Jang, G. Park, J. Cho, and I.-J. Kim. Eldersim: A synthetic data generation platform for human action recognition in eldercare applications. *IEEE Access*, 2021.
- E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. *CoRR*, abs/1605.03170, 2016.
- M. Li, Z. Zhou, J. Li, and X. Liu. Bottom-up pose estimation of multiple person with bounding box constraint. *CoRR*, abs/1807.09972, 2018. URL <http://arxiv.org/abs/1807.09972>.
- G. A. Martínez-Mascorro, J. R. Abreu-Pederzini, J. C. Ortiz-Bayliss, and H. Terashima-Marín. Suspicious behavior detection on shoplifting cases for crime prevention by using 3d convolutional neural networks. *CoRR*, abs/2005.02142, 2020. URL <https://arxiv.org/abs/2005.02142>.
- G. Ning, P. Liu, X. Fan, and C. Zhang. A top-down approach to articulated human pose estimation and tracking. *CoRR*, abs/1901.07680, 2019. URL <http://arxiv.org/abs/1901.07680>.
- G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler, and K. P. Murphy. Towards accurate multi-person pose estimation in the wild. *CoRR*, abs/1701.01779, 2017. URL <http://arxiv.org/abs/1701.01779>.

- L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. V. Gehler, and B. Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. *CoRR*, abs/1511.06645, 2015. URL <http://arxiv.org/abs/1511.06645>.
- X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba. Virtualhome: Simulating household activities via programs. *CoRR*, abs/1806.07011, 2018. URL <http://arxiv.org/abs/1806.07011>.
- D. Rankin, M. Black, R. Bond, J. Wallace, M. Mulvenna, G. Epelde, et al. Reliability of supervised machine learning using synthetic data in health care: Model to preserve privacy for data sharing. *JMIR Medical Informatics*.
- L. Sun, K. Jia, D. Yeung, and B. E. Shi. Human action recognition using factorized spatio-temporal convolutional networks. *CoRR*, abs/1510.00562, 2015. URL <http://arxiv.org/abs/1510.00562>.
- A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, and S. W. Baik. Action recognition in video sequences using deep bi-directional lstm with cnn features. *IEEE Access*, 2018.
- Z. Wu, X. Wang, Y. Jiang, H. Ye, and X. Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. *CoRR*, abs/1504.01561, 2015. URL <http://arxiv.org/abs/1504.01561>.
- Papandreou et al. (2017) Li et al. (2018) Ning et al. (2019) Cao et al. (2016) Pishchulin et al. (2015) Insafutdinov et al. (2016) Fang et al. (2016) Geng et al. (2021) Baccouche et al. (2011) Wu et al. (2015) Sun et al. (2015) Donahue et al. (2014) Gammulle et al. (2017) Ullah et al. (2018) Rankin et al. Chen et al. (2021) Puig et al. (2018) Hwang et al. (2021) Martínez-Mascorro et al. (2020)